RESEARCH

Distributed Stochastic Gradient Descent for Link Prediction in Signed Social Networks

Han Zhang[1], Gang Wu[1] and Qing Ling[2*]

Abstract

This paper considers the link prediction problem defined over a signed social network, where the relationship between any two network users can be either positive (friends) or negative (foes). Given a portion of the relationships, the goal of link prediction is to identify the rest unknown ones. This task resorts to completing the adjacency matrix of the signed social network, which is low-rank or approximately low-rank. Considering the large scale of the adjacency matrix, in this paper we adopt low-rank matrix factorization models for the link prediction problem and solve them through asynchronous distributed stochastic gradient descent algorithms. The low-rank matrix factorization models effectively reduce the size of the parameter space, while the asynchronous distributed stochastic gradient descent algorithms enables fast completion of the adjacency matrix. We validate the proposed algorithms using two real-world datasets on a distributed shared-memory computation platform. Numerical results demonstrate that the asynchronous distributed stochastic gradient descent algorithms achieve nearly linear speedups with respect to the number of computational threads, and are able to complete an adjacency matrix of ten billions of entries within 10 seconds.

Keywords: Signed social network; link prediction; low-rank matrix completion; asynchronous distributed optimization; stochastic gradient descent

1 Introduction

Social network analysis has received much research interest in the past decade. A social network can be regarded as a directed graph, where nodes represent users and weights of edges represent relationships between users. The weight of an edge (i, j) can be either positive, meaning that node i treats node j as a friend, or negative, meaning that node i treats node j as a foe. A case of particular interest is the signed social network, where the weights are either $+1$ or -1. For example, at Epinions, an online review website, a user can choose "trust" or "distrust" for another user [1]. Since collecting all relationships from the users' side is often impossible, estimating the unknown ones is critical to applications such as discovering relationships, recommending friends, identifying malicious users, etc. The target of link prediction over a signed social network is to infer the unknown relationships between users from the known ones.

*Correspondence: lingqing556@mail.sysu.edu.cn

[2] School of Data and Computer Science, Sun Yat-Sen University, No. 132, East Outer Ring Road, 510006, Guangzhou, China

Full list of author information is available at the end of the article

1.1 Related Works

Approaches to solving the link prediction problem over a signed social network can be classified into local, global, and the combination of the two. Below, we give a brief review on these approaches.

Local approaches rely on the theory of structure balance of signed (social) networks, which dates back to 1950s [2]. Structure balance means that the triads in a signed network tend to obey the rules such as "a friend of my friend is my friend", "an enemy of my friend is my enemy", and so on. This observation enables link prediction at a low level. Based on the theory of structure balance, [3] develops a trust (or distrust) propagation framework to predict the trust (or distrust) between pairs of nodes. A logistic regression classifier is proposed in [4], where the features come from only local triad structures of edges. The work of [5] uses additional features that come from longer cycles in a signed network so as to improves the prediction accuracy. Here a longer cycle means a directed path from a node to itself, whose length is longer than that of a triad. However, the longer cycle method increases the number of features exponentially with respect to the length of cycle, and brings difficulties in computation [6]. All these local methods focus on the low-level

structure of a signed network, and ignore the high-level structure. This disadvantage motivates the introduction of global approaches.

Global approaches are built upon the theory of weak balance that characterizes the global structure of a signed (social) network [7]. A complete signed network is weakly balanced, if the nodes can be divided into several clusters such that within-cluster edges are positive and between-cluster edges are negative. It is shown in [8] that a weakly balanced signed network has a low-rank adjacency matrix. Thus, the task of link prediction can be formulated as a low-rank matrix completion problem, and many existing algorithms can be applied as solvers. Two popular low-rank matrix completion models are rank minimization and nuclear norm minimization, where the latter is a convex relaxation of the former [9]. They can be solved by singular value hard thresholding [10] and singular value soft thresholding [11] algorithms, respectively. However, every iteration of these algorithms requires a costly singular value decomposition step on a square matrix, whose numbers of rows and columns are the same as the number of network users. Therefore, they are not suitable for solving large-scale problems. Instead, matrix factorization is a proper model for large-scale low-rank matrix completion, which can be solved by alternating least-squares [12] and stochastic gradient descent [8] algorithms. In [8], the authors demonstrate that stochastic gradient descent is an efficient algorithm to solve the link prediction problem defined over a large-scale signed social network.

Combined approaches utilize both local and global information. For example, [13] proposes a probabilistic matrix factorization method, where each user's feature is determined by its neighbors and social psychology factors. The work of [14] combines features from local triads and global rank minimization to design classification algorithms. The work of [15] builds a probabilistic prediction model, which adaptively adjusts the estimate of adjacency matrix through both local structure balance and global clustering results. However, feature extraction from a large-scale signed social network is often computationally expensive.

This paper focuses on the matrix factorization model for link prediction in a large-scale signed social network, and solves it through the celebrated stochastic gradient descent algorithm. In particular, we take advantages of asynchronous distributed computation to enable fast completion of the large-scale adjacency matrix. On a shard-memory computation platform, Hogwild! distributes stochastic gradient descent steps to multiple computational threads without any locking, and achieves a nearly optimal convergence rate when the optimization problem has a sparse structure. On a master-worker computation platform, [16] proposes the distributed stochastic gradient descent algorithm to for large-scale matrix factorization, which randomly splits the matrix to blocks and assigns different blocks to different workers. To overcome the data discontinuity problem in Hogwild! and the block imbalance and locking problems in distributed stochastic gradient descent, the fixed-point stochastic gradient descent algorithm selects samples within a block in order but randomizes the selection of blocks [17]. Based on Hogwild!, [18] proposes an asynchronous decentralized stochastic gradient descent algorithm, which replaces the global model with a set of local models kept by workers. A decentralized token-based protocol is developed to keep the computation synchronized. In this paper, we shall consider shared-memory computation platform, which fits for distributed implementations of stochastic gradient descent to solve the large-scale low-rank matrix completion problem. We propose two asynchronous distributed stochastic gradient descent algorithms without locking, one is partially asynchronous and close to Hogwild!, while another is fully asynchronous.

1.2 Our Contributions

(i) we consider low-rank matrix factorization models for link prediction of a signed social network, and solve them through asynchronous distributed stochastic gradient descent algorithms. The low-rank matrix factorization models effectively reduce the size of the parameter space, while the asynchronous distributed stochastic gradient descent algorithms enable fast completion of the adjacency matrix.

(ii) We validate the proposed algorithms using two real-world datasets on a distributed shared-memory computation platform. Numerical results demonstrate that the asynchronous distributed stochastic gradient descent algorithms achieve nearly linear speedups with respect to the number of computational threads, and are able to complete an adjacency matrix of ten billions of entries within 10 seconds.

2 Problem Statement

Consider a signed social network with n users, where a user can either like or dislike another user. It can be represented as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, A)$, where $\mathcal{V}$ is the node set whose cardinality is $|\mathcal{V}| = n$, $\mathcal{E}$ is the edge set whose cardinality is $|\mathcal{E}| = m = n^2 - n$, and $A \in \mathbb{R}^{n\times n}$ is the signed adjacency matrix of $\mathcal{G}$. The (i, j)-th entry of A, denoted as a_{ij}, is given by

$$a_{ij} = \begin{cases} 1 & \text{user } i \text{ likes user } j, \\ -1 & \text{user } i \text{ dislikes user } j. \end{cases} \tag{1}$$

With particular note, the diagonal entries a_{ii} are often set to 1, for all $i = 1, \cdots, n$.

The network operator can collect a portion of the relationships from the users' side, but is very likely unable to collect all of them. Denote Ω as the set containing the indexes of the known entries of A. That is, for any $(i, j) \in \Omega$, a_{ij} is known to the network operator. It is of practical importance for the network operator to infer the unknown entries of A from known ones for the sake of discovering relationships of users, recommending friends, identifying malicious users, etc. This link prediction task is essentially a special matrix completion problem, where all the matrix entries are either 1 or -1.

2.1 Rank and Nuclear Norm Minimization Models

The fact that the adjacency matrix of the signed social network is low-rank or approximately low-rank motivates the use of low-rank matrix completion techniques in solving the link prediction problem [8]. A commonly used formulation is

$$\begin{aligned} &\min\ rank(Z), \\ &s.t.\ z_{ij} = a_{ij},\ \forall (i,j) \in \Omega, \\ &\quad z_{ij} \in \{1, -1\},\ \forall i,\ \forall j, \end{aligned} \tag{2}$$

where $Z \in \mathbb{R}^{n\times n}$ is the low-rank matrix to be recovered and $rank(Z)$ denotes the rank of Z. The constraint $z_{ij} = a_{ij}$, $\forall (i, j) \in \Omega$ enforces all known entries in A exactly appear at the corresponding locations in Z. The constraint $z_{ij} \in \{1, -1\}$, $\forall i$, $\forall j$ guarantees that the recovered Z is indeed an adjacency matrix of a signed network. Since $rank(Z)$ is a highly nonconvex cost function, it is often replaced by the nuclear norm of Z, denoted by $\|Z\|_*$. Thus, (2) is relaxed to

$$\begin{aligned} &\min\ \|Z\|_*, \\ &s.t.\ z_{ij} = a_{ij},\ \forall (i,j) \in \Omega, \\ &\quad z_{ij} \in \{1, -1\},\ \forall i,\ \forall j, \end{aligned} \tag{3}$$

Without the discrete constraint $z_{ij} \in \{1, -1\}, \forall i, \forall j$, (2) and (3) can be solved by singular value hard thresholding [10] and singular value soft thresholding [11] algorithms, respectively. Every iteration of these algorithms runs a singular value decomposition step on the intermediate estimate of Z, followed by hard or soft thresholding on the singular values. Since the complexity of singular value decomposition is $O(n^3)$, these algorithms are not applicable to the link prediction problem as n is often large in a social network.

2.2 Matrix Factorization Models

Besides the rank minimization and nuclear minimization models, matrix factorization models are also popular in low-rank matrix completion, especially when the matrix to be completed is large-scale. It is known that any matrix $Z \in \mathbb{R}^{n\times n}$ with rank up to r can be decomposed into a matrix product $Z = X^T Y$, where $X \in \mathbb{R}^{r\times n}$ and $Y \in \mathbb{R}^{r\times n}$. Since the upper bound of the rank is often known or can be estimated in advance, it is natural to introduce a matrix factorization model for link prediction, as

$$\begin{aligned} &\min \sum_{(i,j)\in\Omega} loss(a_{ij}, x_i^T y_j) + \frac{\lambda}{2}\|X\|_F^2 + \frac{\lambda}{2}\|Y\|_F^2, \\ &s.t.\ x_i^T y_j \in \{1, -1\},\ \forall i,\ \forall j. \end{aligned} \tag{4}$$

Here x_i is the i-th column of X, y_j is the j-th column of Y, and $x_i^T y_j$ is the (i, j)-th entry of the matrix to be completed. The distance of a_{ij} and $x_i^T y_j$ is measured by the loss function $loss(\cdot)$. We expect $x_i^T y_j$ to be close to a_{ij} for all $(i, j) \in \Omega$, leading to the minimization of $\sum_{(i,j)\in\Omega} loss(a_{ij}, x_i^T y_j)$. On the other hand, the Frobenius regularization term $\lambda\|X\|_F^2 + \lambda\|Y\|_F^2$, where $\lambda > 0$ is a regularization parameter, is used to avoid overfitting by penalizing the magnitudes of the matrices X and Y.

The matrix factorization model (4) has $2rn$ parameters. In comparison, the rank minimization model (2) and the nuclear norm minimization model (3) both have n^2 parameters. When the matrix to be completed is low-rank, we have $r \ll n$. Therefore, the low-rank matrix factorization models effectively reduce the size of the parameter space. This advantage motivates us to apply the matrix factorization model in solving the link prediction problem defined over a large-scale signed social network.

In the matrix factorization model (4), the loss function $loss(\cdot)$ has several choices. A commonly used one is the square loss

$$loss(a_{ij}, x_i^T y_j) = (a_{ij} - x_i^T y_j)^2, \tag{5}$$

which enforces $x_i^T y_j$ to be as close to a_{ij} as possible, for all $(i, j) \in \Omega$. However, in the link prediction problem, we often allow $x_i^T y_j$ to be biased from a_{ij} in magnitude, and care more about the consistency of their signs. This consideration motivates the introduction of sigmoid loss or squared hinge loss, which penalize the inconsistency of signs [8]. The sigmoid loss and the squared hinge loss are defined as

$$loss_{sigmoid}(a_{ij}, x_i^T y_j) = 1/(1 + \exp(a_{ij} x_i^T y_j)), \tag{6}$$

$$loss_{hinge}(a_{ij}, x_i^T y_j) = (\max(0, 1 - a_{ij} x_i^T y_j))^2, \tag{7}$$

respectively.

3 Method

Based on the matrix factorization model (4), in this section we develop asynchronous distributed stochastic gradient descent algorithms to solve the link prediction problem defined over a large-scale signed social network. We first overview the classical stochastic gradient descent algorithm, and then discuss its asynchronous distributed implementations on a shared-memory computation platform.

3.1 Stochastic Gradient Descent

Consider the cost function of (4). Its full (sub)gradients with respect to x_i and y_j are

$$G_{x_i} = \sum_{j:(i,j)\in\Omega} \frac{\partial loss(a_{ij}, x_i^T y_j)}{\partial x_i} + \lambda x_i,$$
$$G_{y_j} = \sum_{i:(i,j)\in\Omega} \frac{\partial loss(a_{ij}, x_i^T y_j)}{\partial y_j} + \lambda y_j,$$

respectively. Observe that G_{x_i} (or G_{y_j}) are decoupled across i (or j). Thus, a straightforward approach to solving (4) is running a gradient descent algorithm using the full gradients G_{x_i} and G_{y_j} to update all x_i and y_j, followed by projecting the products $x_i^T y_j$ onto the binary set $\{-1, 1\}$. However, every iteration of this full gradient descent algorithm requires to evaluate multiple partial gradients $\partial loss/\partial x_i$ and $\partial loss/\partial y_j$ so as to sum them up. This is time-consuming when the cardinalities of the sets $\{j : (i,j) \in \Omega\}$ and $\{i : (i,j) \in \Omega\}$ are large.

Machine learning theories suggest us to apply the stochastic gradient descent method to handle this issue, through replacing the full gradients by stochastic gradients [19]. To be specific, at every iteration only a random pair of x_i and y_j are updated using their stochastic gradients

$$g_{x_i} = \frac{\partial loss(a_{ij}, x_i^T y_j)}{\partial x_i} + \lambda x_i, \quad (8)$$
$$g_{y_j} = \frac{\partial loss(a_{ij}, x_i^T y_j)}{\partial y_j} + \lambda y_j. \quad (9)$$

Observe that the stochastic gradients g_{x_i} and g_{y_j} are approximations to the full gradients G_{x_i} and G_{y_j}, respectively.

The stochastic gradient descent algorithm has been applied to solve the link prediction problem in [8], as listed in Algorithm 1. The non-negative step size η is often set to be diminishing; otherwise, the gradient noise, which stands for the gap between the full gradient and the stochastic gradient, could prevent the iterate from being convergent.

Algorithm 1 Stochastic Gradient Descent (SGD)

Require: Set Ω, parameter λ
Require: Training set $\{a_{ij}, (i,j) \in \Omega\}$, initial values X and Y
  while not converged **do**
    Uniformly randomly choose $(i,j) \in \Omega$, and pick a step size η
    Calculate stochastic gradients g_{x_i} and g_{y_j} as in (8) and (9)
    Update $x_i \leftarrow x_i - \eta g_{x_i}$ and $y_j \leftarrow y_j - \eta g_{y_j}$
  end while
  For all $(i,j) \notin \Omega$, project $x_i^T y_j$ onto $\{-1, 1\}$

Though the stochastic gradient descent algorithm is proven to be a computationally lightweight approach to solving the link prediction problem, completing a large-scale adjacency matrix is still time-consuming. As we shall see in the numerical experiments, for a signed social network of around 100,000 users, completing the adjacency matrix of ten billion entries by stochastic gradient descent takes a couple of minutes. Thus, for time-sensitive applications, it is necessary to speed up the optimization process. Below we shall develop asynchronous distributed implementations of stochastic gradient descent, which take advantages of multi-thread computing and significantly reduce the running time.

3.2 Distributed Stochastic Gradient Descent

We propose two asynchronous distributed stochastic gradient descent algorithms to solve the link prediction problem defined over a large-scale signed social network. Both algorithms run on a shared-memory system, where training data $(a_{ij}, (i,j) \in \Omega)$ and optimization variables (x_i and y_j, $\forall i$, $\forall j$) are stored in the memory, while multiple computational threads asynchronously pick up the training data and the optimization variables, perform stochastic gradient descent, and save the updated optimization variables in the memory. This shared-memory structure is particularly fit for the link prediction task due to its high throughput and low latency. In our workstation whose detailed specifications shall be given in Section 4, each computational thread can read and write a contiguous physical memory at a speed of over 4GB/s with latency in tens of nanoseconds, and ordinal multiple hard disks can load data into the memory at a speed of over 500MB/s. Since each stochastic gradient descent step for link prediction is also lightweight, the reading, computation and writing procedures are all efficient. In comparison, if we use the master-worker structure that is also popular in distributed computing, each master shall store a portion of the training data and handle a portion of the optimization variables. According to the stochastic gradient descent algorithm for link prediction, in this case masters have to frequently read training data from others, which is slow in the master-worker structure.

A key common feature of the two proposed asynchronous distributed stochastic gradient descent algorithms is *lock-free updating*, which is also adopted in Hogwild! [20]. To be specific, when a computational thread is assigned to update x_i and y_j, it reads a_{ij} and the current x_i and y_j from the memory to run a stochastic gradient step, followed by writing the updated x_i and y_j to the memory, even when another computational thread is using the current x_i or y_i. This inconsistency in updating the same optimization variable wastes the computational power and may cause errors, especially when the number of computational threads increases. Fortunately, to solve the link prediction problem, each stochastic gradient descent step only modifies a very small part of the optimization variables, such that memory overwrites are rare. Therefore, the lock-free scheme remains to be efficient due to the sparse data access pattern.

The proposed distributed algorithms are partially asynchronous and fully asynchronous, as outlined in Algorithm 2 and 3, respectively. Their difference is as follows. In the partially asynchronous algorithm, computations are split into rounds and every round processes all the training data indexed by Ω. Within each round, every computational thread handles one split of the training data in an asynchronous manner. In comparison, the fully asynchronous algorithm does not have the concept of "round". The computational threads simply sample from Ω and run stochastic gradient descent steps without any coordination.

Suppose that there are M computational threads. In the partially asynchronous distributed stochastic gradient algorithm, the shared memory stores the training set $\{a_{ij}, (i,j) \in \Omega\}$, as well as the initial values of X and Y. Any one of the M computational threads, which we number as 0, knows the set Ω. The parameter λ is known to all the computational threads. In the beginning of every round, computational thread 0 permutes the elements in Ω, equally divides the elements into M subsets $\Omega_1, \cdots, \Omega_M$, and assigns them to computational threads $m = 1, \cdots, M$, respectively. After that, the computational threads handle their training subsets asynchronously. Computational thread m chooses index $(i,j) \in \Omega_m$ in order, read the training sample a_{ij} and the current values of x_i and y_j from the shared memory, and then calculates the stochastic gradients g_{x_i} and g_{y_j} as in (8) and (9). With x_i, y_j, g_{x_i} and g_{y_j} at hand, computational thread m picks a diminishing step size η, updates $x_i \leftarrow x_i - \eta g_{x_i}$ and $y_j \leftarrow y_j - \eta g_{y_j}$ through stochastic gradient descent, and then writes the new x_i and y_j to the shared memory. When a certain stopping criterion is met, computational thread 0 projects $x_i^T y_j$ onto $\{-1, 1\}$ for all $(i,j) \notin \Omega$, which are the estimates to the unknown elements of the adjacency matrix.

Algorithm 2 Partially Asynchronous Distributed SGD

Require: Set Ω, parameter λ
Require: Training set $\{a_{ij}, (i,j) \in \Omega\}$, initial values X and Y
  while not converged **do**
    Computational thread 0
    Permute elements in Ω
    Equally divide the elements to M subsets $\Omega_1, \cdots, \Omega_M$
    Computational thread $m = 1, \cdots, M$ **asynchronously**
    Choose $(i,j) \in \Omega_m$ in order
    Read a_{ij}, x_i and y_j from memory, and pick a step size η
    Calculate stochastic gradients g_{x_i} and g_{y_j} as in (8) and (9)
    Update $x_i \leftarrow x_i - \eta g_{x_i}$ and $y_j \leftarrow y_j - \eta g_{y_j}$
    Write x_i and y_j to memory
  end while
  Computational thread 0
  For all $(i,j) \notin \Omega$, project $x_i^T y_j$ onto $\{-1, 1\}$

In the fully asynchronous distributed stochastic descent algorithm, the shared memory stores the training set $\{a_{ij}, (i,j) \in \Omega\}$, as well as the initial values of X and Y. The index set Ω and the parameter λ are known to all the computational threads, which asynchronously run stochastic gradient descent steps. Every computational thread m uniformly randomly chooses an index from the full index set, namely, $(i,j) \in \Omega$. Then, it reads a_{ij}, x_i and y_j, calculates stochastic gradients g_{x_i} and g_{y_j}, updates x_i and y_j, and then writes x_i and y_j to the memory. This procedure repeats when a certain stopping criterion is met, without any synchronous coordination. Finally, one of the computational threads, which is numbered as 0, projects $x_i^T y_j$ onto $\{-1, 1\}$ for all $(i,j) \notin \Omega$.

The two proposed asynchronous algorithms differ in the way of using the training data. In every round of the partially asynchronous algorithm, the computational threads use all of the training data. In the fully asynchronous algorithm, it is possible that one training sample is not used while another one has been used for several times. The partially asynchronous algorithm may waste time in the synchronization step, if the computational threads have imbalanced computational powers, or the training set is not evenly partitioned. These issues can be addressed if the shared-memory system is properly configured. On the other hand, the imbalanced use of training samples may delay the optimization process of the fully asynchronous algorithm. Nevertheless, when the training data are large-scale, we observe that the fully asynchronous algorithm performs well. Indeed, both algorithms demonstrate nearly linear speedup with respect to the number of computational threads, as we shall show in the numerical experiments.

4 Results And Discussion

In the numerical experiments we validate the effectiveness of the proposed asynchronous distributed stochastic gradient descent algorithms in link prediction on

Algorithm 3 Fully Asynchronous Distributed SGD

Require: Set Ω, parameter λ
Require: Training set $\{a_{ij}, (i,j) \in \Omega\}$, initial values X and Y
  while not converged **do**
    Computational thread $m = 1, \cdots, M$ **asynchronously**
    Uniformly randomly choose $(i,j) \in \Omega$
    Read a_{ij}, x_i and y_j from memory, and pick a step size η
    Calculate stochastic gradients g_{x_i} and g_{y_j} as in (8) and (9)
    Update $x_i \leftarrow x_i - \eta g_{x_i}$ and $y_j \leftarrow y_j - \eta g_{y_j}$
    Write x_i and y_j to memory
  end while
  Computational thread 0
  For all $(i,j) \notin \Omega$, project $x_i^T y_j$ onto $\{-1, 1\}$

two real-world datasets, Slashdot and Epinions [1]. The algorithms are coded in C++ and run in a shared-memory system on a workstation using pthread. The workstation has a 128GB RAM, a 3.2TB hard disk and a dual Xeon E5-2630 CPUs. Each CPU has 10 cores and each core has maximally 2 computational threads. Thus, the maximum number of computational threads is 40.

4.1 Datasets

Slashdot is a technology-related news platform. Users in Slashdot can tag each other as friend (positive) or foe (negative). They can also write news articles and other users make comments on these articles. **Epinions** is a who-trust-whom consumer review site, in which users can decide to trust (positive) or distrust (negative) other users. They can also post reviews for products and other users express opinions on these reviews with their ratings.

Table 1 shows the statistics of these two datasets, from which we observe that: (i) the matrices to be completed are large-scale, with billions of entries; (ii) the known entries in the adjacency matrices are very sparse and most of the known edges are positive.

Table 1 Statistics of the Slashdot and Epinions Datasets

	Slashdot	Epinions
# of Users	77357	131828
# of Known Edges	516575	841372
% of Positive Known Edges	76.7%	85.3%
% of Negative Known Edges	23.3%	14.7%

4.2 Experimental Settings

In the numerical experiments, we compare the following three algorithms: stochastic gradient descent in [8], partially asynchronous distributed stochastic gradient descent and fully asynchronous distributed stochastic gradient descent. For the two asynchronous distributed algorithms proposed in this paper, we vary the number of computational threads to quantify their speedups. Particularly, when the number of computational threads is 1, the fully asynchronous distributed stochastic gradient descent algorithm reduces to the classic stochastic gradient descent algorithm. In the matrix factorization model (4), we test the square loss in (5), the sigmoid loss in (6) and the squared hinge loss in (7).

The performance of the three algorithms is evaluated by the prediction accuracy. We randomly sample 90% of the data as the training set and the remaining 10% as the test set. The estimates are compared with the values in the test set, and the accuracy is defined as the percentage of the correct estimates. We run all the experiments for 10 times and average the results to ensure reliability.

In the numerical experiments, the initial values of X and Y are randomly generated and every entry follows a standard Gaussian distribution. The step size η is set to be $\eta_0/(1+k)^{0.1}$, where η_0 is the initial step size and hand tuned to be the best, and k is the number of epochs. By one epoch, we mean that $|\Omega|$ samples have been used during that period. Note that in the partially asynchronous distributed algorithm, one epoch corresponds to one round where every sample is used for one time. However, in one epoch of the classic stochastic gradient descent algorithm and the fully asynchronous distributed algorithm, some samples may be used for more than one time, while some other samples may be never used. We vary the estimated rank r of the adjacency matrix and the regularization parameter λ in the numerical experiments.

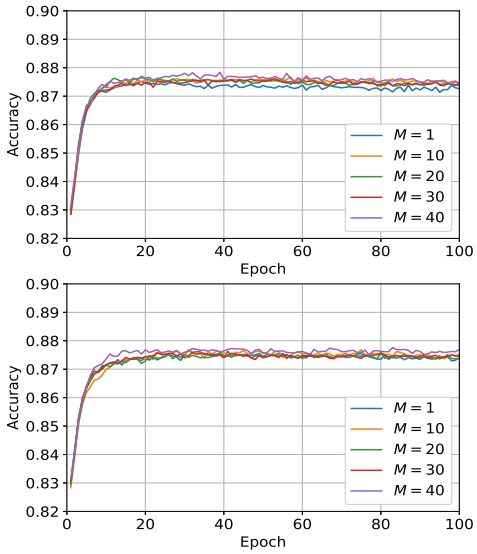

Figure 1 Accuracy on Slashdot using sigmoid loss. TOP: partially asynchronous. Bottom: fully asynchronous.

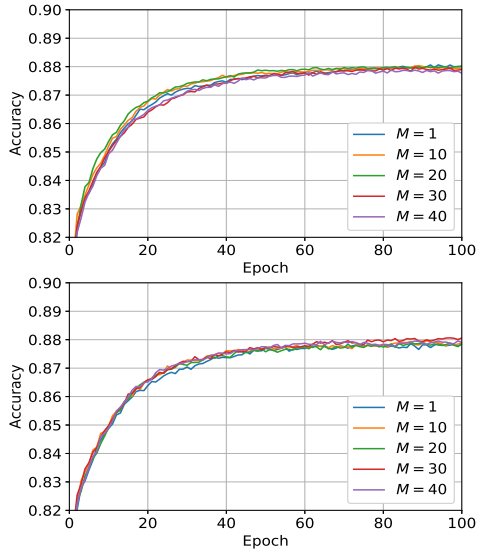

Figure 2 Accuracy on Slashdot using squared hinge loss. TOP: partially asynchronous. Bottom: fully asynchronous.

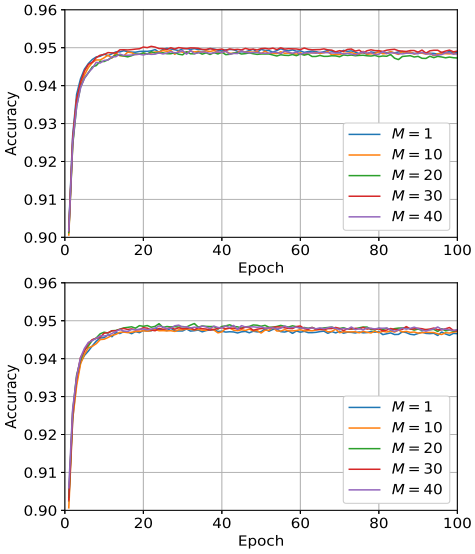

Figure 4 Accuracy on Epinions using sigmoid loss. TOP: partially asynchronous. Bottom: fully asynchronous.

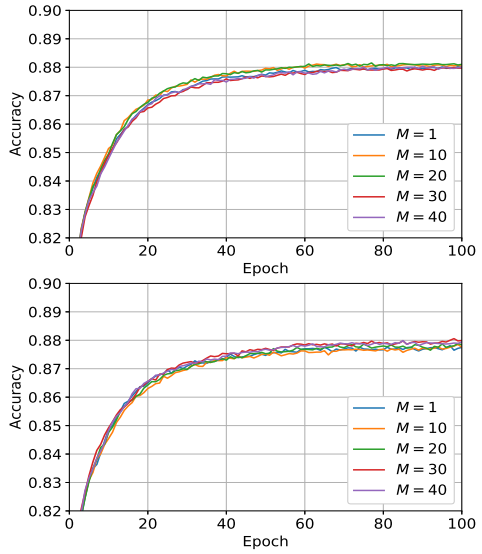

Figure 3 Accuracy on Slashdot using square loss. TOP: partially asynchronous. Bottom: fully asynchronous.

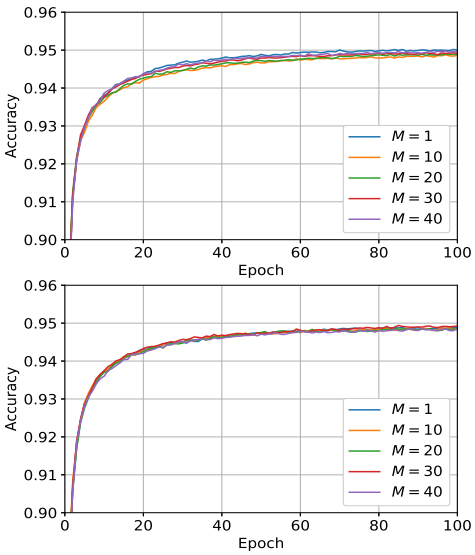

Figure 5 Accuracy on Epinions using squared hinge loss. TOP: partially asynchronous. Bottom: fully asynchronous.

4.3 Comparisons on Accuracies

Figures 1, 2 and 3 depict the prediction accuracies of the algorithms on the Slashdot dataset using sigmoid ($\lambda = 0.01$), squared hinge ($\lambda = 0.4$) and square ($\lambda = 0.4$) losses, respectively. Observe that as long as the number of epochs is fixed, increasing the number

of computational threads does not change the accuracy too much. The partially asynchronous and fully asynchronous algorithms also perform similarly. After 100 epochs, the three losses yield similar accuracies, though the curves of the sigmoid loss are not as steady as those of the square loss and the squared hinge loss.

On the Epinions dataset, we also show the prediction accuracies using sigmoid ($\lambda = 0.01$), squared hinge ($\lambda = 0.4$) and square ($\lambda = 0.4$) losses in Figures 4, 5 and 6, respectively. Similar to the results of the Slashdot dataset, the prediction accuracies with respect to the number of epochs are not sensitive in the partially or fully asynchronous algorithms, the number of computational threads, as well as the choice of loss function. Different to the results of the Epinions dataset, the curves of the squared hinge loss and the squared loss are steady, while those of the sigmoid loss are slightly fluctuant. The steady-state prediction accuracy of the Epinions dataset is around 0.945, higher than that of the Slashdot dataset, which is around 0.875. Our conjecture is that, the percentage of positive known edges in the Slashdot dataset is 76.7% and that in the Epinions dataset is 85.3%, such that estimating an unknown edge to be positive is a quite safe choice in the Epinions dataset. Therefore, it is relatively easier to achieve a higher prediction accuracy in the Epinions dataset.

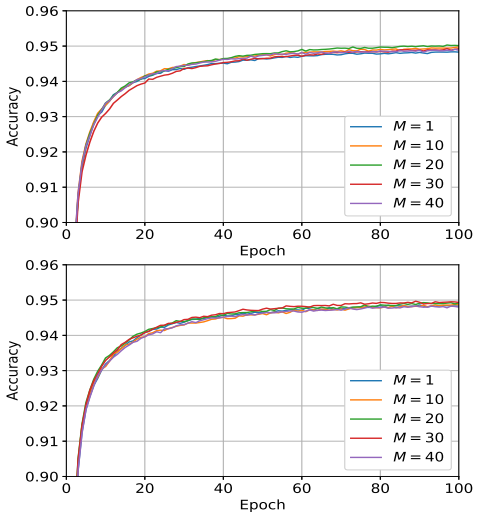

Figure 6 Accuracy on Epinions using square loss. TOP: partially asynchronous. Bottom: fully asynchronous.

4.4 Rank Estimate and Regularization Parameter

Now we analyze the sensitivity of the proposed asynchronous distributed algorithms to the rank estimate and the regularization parameter. We only consider the squared hinge loss in this set of experiments.

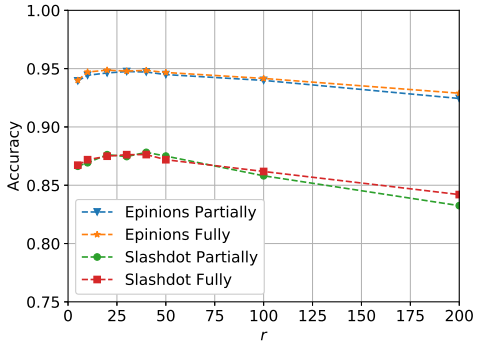

Figure 7 Sensitivity to the rank estimate r. The loss function is squared hinge, the number of computational threads is $M = 10$, and the regularization parameter is $\lambda = 0.2$.

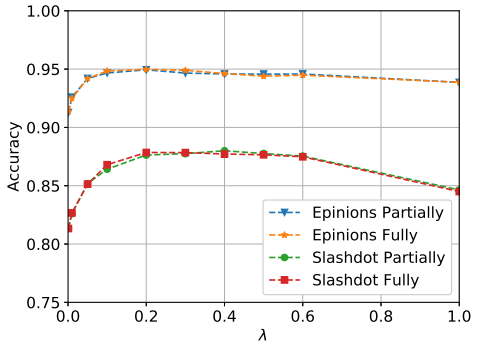

Figure 8 Sensitivity to the regularization parameter λ. The loss function is squared hinge, the number of computational threads is $M = 10$, and the rank estimate is $r = 30$.

In Figure 7, we show the impact of rank estimate r on the prediction accuracy. The number of computational threads is $M = 10$, and the regularization parameter is $\lambda = 0.2$. For both partially and fully asynchronous algorithms and both Slashdot and Epinion datasets, the prediction accuracies are not sensitive to the choice of r, and $r \simeq 30$ yields the best results. This makes sense because a small r is insufficient to recover the high-dimensional adjacency matrix, while a large r lacks the generalization ability.

As depicted in Figure 8, the prediction accuracy is also insensitive to the regularization parameter λ. A moderate value of $\lambda \simeq 0.2$ helps both algorithms reach the best accuracies in both datasets.

4.5 Nearly Linear Speedups

In the last set of the numerical experiments, we shall show that the proposed asynchronous distributed algo-

rithms are computationally efficient, in the sense that they achieve nearly linear speedups with respect to the number of computational threads.

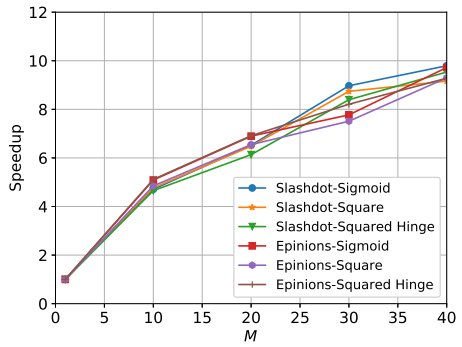

Figure 9 Speedup of the partially asynchronous distributed stochastic gradient descent algorithm.

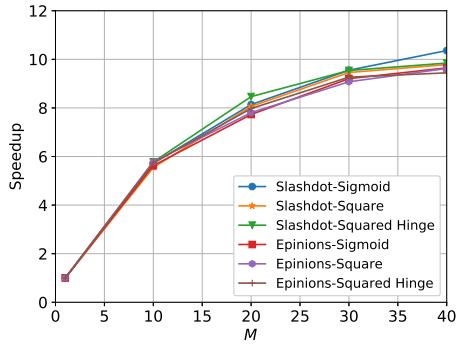

Figure 10 Speedup of the fully asynchronous distributed stochastic gradient descent algorithm.

Figures 9 and 10 demonstrate the speedups of the partially and fully asynchronous distributed stochastic gradient descent algorithms, respectively. In both datasets, both algorithms are able to achieve nearly linear speedups. When the number of computational threads is 10, the speedup is around 5 for the partially asynchronous algorithm and around 6 for the fully asynchronous algorithm. When the number of computational threads is 40, the speedup becomes around 9 and around 10 for the partially and fully asynchronous algorithms, respectively. The Hogwild! style partially asynchronous distributed stochastic gradient descent algorithm requires coordination before every round, which results in the additional cost in terms of computational time. In comparison, the fully asynchronous distributed stochastic gradient descent algorithm, though may suffer from the imbalanced use of training samples, still performs well for over the large-scale datasets.

Table 2 gives the running times of the asynchronous distributed algorithms for 100 epochs on the two datasets. When the number of computational threads is $M = 1$, the fully asynchronous distributed stochastic gradient descent algorithm degenerates to the classic stochastic gradient descent algorithm, which needs around 1 minute on the Slashdot dataset and around 2 minutes on the Epinions dataset. In comparison, when the number of computational threads increases to $M = 40$, the running times are around 7 seconds and 10 seconds, respectively. These significant savings show that the proposed algorithms are particularly fit for time-sensitive applications.

5 Conclusion

This paper proposes two asynchronous distributed stochastic gradient descent algorithms to solve the link prediction problem defined over a large-scale signed-network. The link prediction problem is formulated a matrix factorization model, which aims to complete the low-rank adjacency matrix. The two proposed distributed stochastic gradient descent algorithms, one is fully asynchronous and another is partially asynchronous, are shown to be powerful tools to solve this problem on a shared-memory computation platform. Numerical experiments on two real-world large-scale datasets demonstrate that the two proposed asynchronous distributed algorithms have nearly linear speedups, and are able to complete an adjacency matrix of ten billions of entries within 10 seconds.

Abbreviations

SGD: Stochastic gradient desecnt; RAM: Random access memory; CPU: Central processing unit

Declarations

Availability of data and material

Source codes are available upon request. Datasets are available on http://snap.stanford.edu

Competing interests

The authors declare that they have no competing interests.

Funding

Qing Ling is supported in part by the NSF China under Grant 61573331 and the NSF Anhui under Grant 1608085QF130,

Author's contributions

Han Zhang developed the algorithm, implemented the algorithm, and drafted the paper. Gang Wu revised the paper. Qing Ling developed the algorithm and drafted the paper.

Acknowledgements

Not available.

Author details

[1] Department of Automation, University of Science and Technology of China, No. 443, Huangshan Road, 230027, Hefei, China. [2] School of Data and Computer Science, Sun Yat-Sen University, No. 132, East Outer Ring Road, 510006, Guangzhou, China.

Table 2 Running Times of the Asynchronous Distributed Algorithms for 100 Epochs in Seconds

	Partially Asynchronous			Fully Asynchronous		
	Sigmoid	Square	Squared hinge	Sigmoid	Square	Squared hinge
Slashdot, M = 1	72.66	68.44	63.80	70.97	68.38	64.59
Slashdot, M = 10	15.43	14.35	13.71	12.40	12.33	11.18
Slashdot, M = 20	11.14	10.56	10.40	8.72	8.48	7.63
Slashdot, M = 30	8.10	7.83	7.60	7.43	7.23	6.77
Slashdot, M = 40	7.42	7.47	6.69	6.85	6.99	6.56
Epinions, M = 1	118.61	108.68	104.60	112.40	108.65	103.60
Epinions, M = 10	23.30	22.41	20.44	19.98	18.79	18.05
Epinions, M = 20	17.20	16.60	15.13	14.54	13.92	12.99
Epinions, M = 30	15.26	14.47	12.74	12.21	11.97	11.18
Epinions, M = 40	12.20	11.69	11.27	11.65	11.29	10.97

References

1. Leskovec, J., Huttenlocher, D., Kleinberg, J.: Signed networks in social media. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 1361–1370 (2010)
2. Cartwright, D., Harary, F.: Structural balance: A generalization of Heider's theory. Psychological Review **63**(5), 277–293 (1956)
3. Guha, R., Kumar, R., Raghavan, P., Tomkins, A.: Propagation of trust and distrust. In: Proceedings of the 13th International Conference on World Wide Web, pp. 403–412 (2004)
4. Leskovec, J., Huttenlocher, D., Kleinberg, J.: Predicting positive and negative links in online social networks. In: Proceedings of the 19th International Conference on World Wide Web, pp. 641–650 (2010)
5. Chiang, K.-Y., Natarajan, N., Tewari, A., Dhillon, I.S.: Exploiting longer cycles for link prediction in signed networks. In: Proceedings of the 20th ACM International Conference on Information and Knowledge Management, pp. 1157–1162 (2011)
6. Chiang, K.-Y., Hsieh, C.-J., Natarajan, N., Dhillon, I.S., Tewari, A.: Prediction and clustering in signed networks: A local to global perspective. Journal of Machine Learning Research **15**(1), 1177–1213 (2014)
7. Davis, J.A.: Clustering and structural balance in graphs. Human Relations **20**(2), 181–187 (1967)
8. Hsieh, C.-J., Chiang, K.-Y., Dhillon, I.S.: Low rank modeling of signed networks. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 507–515 (2012)
9. Recht, B., Fazel, M., Parrilo, P.A.: Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. SIAM Review **52**(3), 471–501 (2010)
10. Goldfarb, D., Ma, S.: Convergence of fixed-point continuation algorithms for matrix rank minimization. Foundations of Computational Mathematics **11**(2), 183–210 (2011)
11. Cai, J., Candes, E.J., Shen, Z.: A singular value thresholding algorithm for matrix completion. SIAM Journal on Optimization **20**(4), 1956–1982 (2010)
12. Koren, Y., Bell, R.M., Volinsky, C.: Matrix factorization techniques for recommender systems. IEEE Computer **42**(8), 30–37 (2009)
13. You, Q., Wu, O., Luo, G., Hu, W.: A probabilistic matrix factorization method for link sign prediction in social networks. In: Machine Learning and Data Mining in Pattern Recognition, pp. 15–420 (2016)
14. Shahriari, M., Sichani, O.A., Gharibshah, J., Jalili, M.: Sign prediction in social networks based on users reputation and optimism. Social Network Analysis and Mining **6**(1), 91 (2016)
15. Javari, A., Qiu, H., Barzegaran, E., Jalili, M., Chang, K.C.-C.: Statistical link label modeling for sign prediction: Smoothing sparsity by joining local and global information. In: Proceedings of the 17th IEEE International Conference on Data Mining, pp. 1039–1044 (2017)
16. Gemulla, R., Nijkamp, E., Haas, P.J., Sismanis, Y.: Large-scale matrix factorization with distributed stochastic gradient descent. In: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 69–77 (2011)
17. Chin, W., Zhuang, Y., Juan, Y., Lin, C.: A fast parallel stochastic gradient method for matrix factorization in shared memory systems. ACM Transactions on Intelligent Systems and Technology **6**(1), 2 (2015)
18. Zhang, H., Hsieh, C.-J., Akella, V.: Hogwild++: A new mechanism for decentralized asynchronous stochastic gradient descent. In: Proceedings of the 16th IEEE International Conference on Data Mining, pp. 629–638 (2016)
19. Bottou, L.: Large-scale machine learning with stochastic gradient descent. In: Proceedings of 19th International Conference on Computational Statistics, pp. 177–186 (2010)
20. Recht, B., Re, C., Wright, S., Niu, F.: Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In: Advances in Neural Information Processing Systems, pp. 693–701 (2011)

Figure legend

Fig. 1: Accuracy on Slashdot using sigmoid loss. TOP: partially asynchronous. Bottom: fully asynchronous. ($M = 1$, $M = 10$, $M = 20$, $M = 30$, $M = 40$)

Fig. 2: Accuracy on Slashdot using squared hinge loss. TOP: partially asynchronous. Bottom: fully asynchronous. ($M = 1$, $M = 10$, $M = 20$, $M = 30$, $M = 40$)

Fig. 3: Accuracy on Slashdot using square loss. TOP: partially asynchronous. Bottom: fully asynchronous. ($M = 1$, $M = 10$, $M = 20$, $M = 30$, $M = 40$)

Fig. 4: Accuracy on Epinions using sigmoid loss. TOP: partially asynchronous. Bottom: fully asynchronous. ($M = 1$, $M = 10$, $M = 20$, $M = 30$, $M = 40$)

Fig. 5: Accuracy on Epinions using squared hinge loss. TOP: partially asynchronous. Bottom: fully asynchronous. ($M = 1$, $M = 10$, $M = 20$, $M = 30$, $M = 40$)

Fig. 6: Accuracy on Epinions using square loss. TOP: partially asynchronous. Bottom: fully asynchronous. ($M = 1$, $M = 10$, $M = 20$, $M = 30$, $M = 40$)

Fig. 7: Sensitivity to the rank estimate r. The loss function is squared hinge, the number of computational threads is $M = 10$, and the regularization parameter is $\lambda = 0.2$. (Epinions Partially, Epinions Fully, Slashdot Partially, Slashdot Fully)

Fig. 8: Sensitivity to the regularization parameter λ. The loss function is squared hinge, the number of computational threads is $M = 10$, and the rank estimate is $r = 30$. (Epinions Partially, Epinions Fully, Slashdot Partially, Slashdot Fully)

Fig. 9: Speedup of the partially asynchronous distributed stochastic gradient descent algorithm. (Slashdot-Sigmoid, Slashdot-Square, Slashdot-Squared Hinge, Epinions-Sig, Epinions-Square, Epinions-Squared Hinge)

Fig. 10: Speedup of the fully asynchronous distributed stochastic gradient descent algorithm. (Slashdot-Sigmoid, Slashdot-Square, Slashdot-Squared Hinge, Epinions-Sig, Epinions-Square, Epinions-Squared Hinge)